Corpus-based stochastic finite-state predictive text entry for reduced keyboards: application to Catalan

Mikel L. Forcada
Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant,
E-03071 Alacant, Spain
http://www.dlsi.ua.es/~mlf/

Abstract Users of digital mobile phones have access to an increasing number of text-based services but most of them can only use a very reduced keyword for text entry. Traditionally, this has been solved by means of multiple tapping and delays or next-character keys (e.g. 66(pause)666 for no), which is slow (two taps per letter on average), inconvenient, and prone to error. Recently, some mobile terminals offer dictionary-based *predictive text-entry* schemes which, in almost all cases, allow for one-tap-per-letter text entry (e.g. 367 for for). This paper shows how text corpora can be used to build stochastic finite-state automata which may in turn be easily used to implement predictive text entry for medium-sized languages such as Catalan, not currently supported by major predictive-text-entry companies.

1 Introduction

Users of digital mobile phones have access to an increasing number of text-based services: short messages (SMS), chat, internet browsing via the wireless access protocol (WAP), etc., but most of them can only use a very reduced keyword (about a dozen keys) for text entry. Figure 1 sketches a typical mobile phone keyboard. Traditionally, text entry has been solved by means of multiple tapping (e.g. 222 for 'c') and 1-second delays or next-character keys (e.g. 66(pause or next-character key)666 for 'no'), which is slow, inconvenient, and prone to error. The average number of keypresses (taps) necessary to write a message varies with language but ranges around two.[1] More recently, medium- and high-end mobile phone terminals offer dictionary-based *predictive text-entry* systems which, in almost all cases, allow for one-tap-per-letter text entry (e.g. 367 for 'for' with an English dictionary). There are a number of companies licensing such systems to manufacturers of mobile phones: Tegic Communications' *T9* (http://www.t9.com, see also Kushler (1998)) and Zi Corporation's *eZiText* (http://www.zicorp.com).[2] The operation of these two systems is slightly different: whereas *T9* shows only the most likely prefix (or whole word) for the sequence keyed so far, *eZiText* tries to provide a number of very likely completions to it which pop up and can be selected using a special key. This paper shows how the statistics in a text corpus can be easily used to build a stochastic finite-state automaton which may be easily used to implement predictive text entry —with word completion— for medium-sized languages such as Catalan (6 million speakers), which are not currently supported by major predictive text-entry companies.[3] Currently, predictive-text-entry dictionaries and software are stored as firmware in mobile phones, but recently mobile phones are capable of using soft-

[1] Obviously, if no mistake is made.

[2] There are two additional fast text entry system, Eatoni's *wordwise* and *letterwise* (http://www.eatoni.com), but, to my knowledge, these are not dictionary-based, but rather rely on letter distributions.

[3] Eatoni do offer Catalan among other medium-sized languages but their method is based on letter statistics, not on dictionaries. The use of text statistics to implement disambiguation strategies for reduced keyboards is not new, see Arnott and Javed (1992).

ware and data downloaded from the network itself. Note that the actual technology used by *T9*'s and *eZiText*'s proprietary dictionary-based predictive text entry has not been disclosed and may differ from the one described here, although it is very unlikely to be very different.

This paper is organized as follows. Section 2 describes the use of corpus-based finite-state stochastic dictionaries; section 3 describes how they may be applied to predictive text entry in a mobile telephone, both from the user's side and from the system side. Section 4 describes simulation experiments with a Catalan corpus-based finite-state predictive-text-entry system. Finally, concluding remarks are given in section 5.

2 Finite-state stochastic dictionaries from corpora

The key ingredient in dictionary-based predictive-text-entry systems may be modelled as a stochastic finite-state machine; in particular, a trie with probabilities corresponding to frequencies observed for words and their prefixes in suitable corpora. These machines may be made more compact by merging those states having similar continuation statistics, using techniques such as the ones developed by Carrasco and Oncina (1994); this is very important when one considers that memory saving is at a premium in handheld devices, especially when updates or data have to be downloaded through the network at low rates. Results reported here are those obtained without performing this compaction.

A *finite-state stochastic dictionary* is a 6-tuple

$$M = (Q, \Sigma, \delta, p, q_I, \pi), \tag{1}$$

where $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ is a set of finite states, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ is the alphabet of symbols used in words, $\delta : Q \times \Sigma \to Q$ is the next-state function, $p : Q \times \Sigma \to [0, 1]$ assigns probability to those transitions possible according to δ, q_I is the initial state, $\pi : Q \to [0, 1]$ assigns nonzero termination probability to those states corresponding to whole words.

In particular, if $W \in \Sigma^*$ is the susbet of words present, states represent all the possible prefixes of words in W, that is, $Q = \Pr(W)$ where $\Pr(W) = \{z \in \Sigma^* : (\exists x \in \Sigma^* : zx \in W)\}$; the next-state function δ is therefore trivially defined as follows: $\delta(q, a) = qa$ for all $q \in Q$ and for all $a \in \Sigma$ such that $qa \in Q$, and undefined otherwise. When a corpus S is used, the relative frequencies of words $f_S : W \to [0, 1]$ (after suitable removal of *hapax legomena* or smoothing) may be used to determine p and π as follows. The values of p for all $q \in Q$ and all $a \in \Sigma$ are estimated as

$$p(q, a) = \frac{c_S(qa)}{c_S(q)} \tag{2}$$

where

$$c_S(q) = \sum_{z \in \Sigma^* : qz \in W} f_S(qz) \tag{3}$$

are the frequencies of prefixes. The values of the termination probabilities π are estimated for all $q \in Q$ as

$$\pi(q) = \frac{f_S(q)}{c_S(q)} \quad \forall q \in Q. \tag{4}$$

Although it may be easily computed by traversing the finite-state stochastic dictionary, it may be convenient (if memory is not scarce) to store with each state (prefix) in Q the continuation or suffix that turns it into the most likely whole word.[4] If function $C : Q \to \Sigma^*$ assigns a continuation string to all states in Q, the winning continuation of a prefix q is given by

$$C(q) = \text{argmax}_{z \in \Sigma^*} f_S(qz). \tag{5}$$

3 Using finite-state dictionaries for predictive text entry

3.1 The keyboard

Let K be the alphabet of those keys $K = \{1, 2, \ldots, 9\}$ onto which letters (or symbols making up words) are mapped (see fig. 1), and $\Sigma_k \subset \Sigma$ the set of letters mapped to key $k \in K$. Table 1 shows a possible assignment of Catalan letters and symbols to keys.

[4] Storing continuations has been the choice in the simulations below.

1 ' - ·	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUV	9 WXYZ
* *next*	0 .	# *space*

Figure 1: A mobile phone keypad

KEY	SYMBOLS
1	· - ’
2	a b c à ç
3	d e f è é
4	g h i ï
5	j k l
6	m n o ó ò
7	p q r s
8	t u v ú ü
9	w x y z

Table 1: Assignment of Catalan letters and symbols to keys. In key 1, the centered dot ‘·’ is used as part of the trigraph ‘l·l’ representing a double ‘l’ sound; apostrophes and hyphens are used to attach articles, prepositions and object pronouns to words: *l’autòmat*, *Universitat d’Alacant*, *programar-lo*.

3.2 User interface

From the user's side, the system works as follows. Assume that the user wants to key in the word “barcelona”.[5] The user may blindly —that is, by looking at the keyboard, not at the screen— key in the sequence of keys containing these letters, 227235662, then check whether the screen shows `barcelona` as desired, and validate it by pressing the space (`#`) key (this is similar to the *modus operandi* of *T9*). If the word on the screen is not the one desired, the user can press the NEXT (`*`) key repeatedly until the desired word shows on the screen. For example, if the user wants the word `cara`, and types 2272, the system may show `barc-`, the most likely prefix (with the hyphen indicating that it does not correspond to a whole word); after pressing `*` three times, `cara` will be shown on the screen so that pressing `#` will validate it. But a “word completion” feature similar to that found in *eZiText* will also be supported. It will be activated by using an additional key (a typical choice would be a function key such as the right-arrow cursor key). When the user has keyed in 2272, the system shows `barc-` and the most likely completion somewhere else in the screen (`barcelona`); if this completion is the one desired, the user can simply hit a suitable completion key to validate the whole word, which has taken only 5 keystrokes to produce. However, the user has to check the screen for suggested completions, which requires substantial additional attention.

There are two special situations that may happen with words not found in the dictionary. One is when the user introduces a sequence of keys for which no valid prefix can be found in the dictionary; in this case, a beep will be heard and the system may fall back to “traditional” multitap keying. The other situation is when the user keys in the whole sequence, hears no beep, but does not find the word after repeatedly hitting the NEXT key, `*`; in that case, hitting the delete key will erase the whole word —*T9* erases only the last letter— and the system will enter the traditional mode for that particular word (this can be improved to save some work but does not happen very often).

3.3 Computational issues

From the system's side, the computation is easy: for each key sequence, a list of candidate prefixes is easily built. The construction may be defined recursively as follows. Let K^* be the set of all finite-length sequences of keypresses from K. Let us denote the function computing the list of candidate prefixes for a key sequence as $\text{cand} : K^* \to 2^Q$, so that the set of candidate prefixes for sequence $v \in K^*$ is $\text{cand}(v)$. Clearly,

$$\text{cand}(\epsilon) = \{\epsilon\} \tag{6}$$

[5] In this paper, the introduction of uppercase letters will not be considered; this feature can easily be implemented. *T9* uses repeated strokes on the space key `#` to select all lowercase, all uppercase, or first letter uppercase and the rest lowercase.

where ϵ stands for the empty string, and

$$\text{cand}(vk) = \{x\sigma \in Q : x \in \text{cand}(v), \sigma \in \Sigma_k\}. \quad (7)$$

If $\text{cand}(vk) = \emptyset$ for a given keypress sequence v, a beep is produced and the system stores the list $\text{cand}(v)$, so that one of the prefixes may be selected and continued using the traditional (multi-tap) method. Candidate prefixes are sorted according to their probability, so that the most probable candidate is shown first and the key * may be used to select the second most probable, the third, etc. Probability is also recursively computed for each candidate prefix as follows:

$$P(\epsilon) = 1, \quad (8)$$

$$P(q\sigma) = P(q)p(q, \sigma). \quad (9)$$

If the current prefix q is not a complete word (that is, $\pi(q) = 0$), it is shown with a dangling hyphen. No bias in favor of showing complete forms if present has been used (Tegic's *T9* seems to do so).

4 Experiments

4.1 The corpus

As in the work of Dunlop and Crossan (1999) and Dunlop and Crossan (2000), a text corpus of about 900,000 words corresponding to November 2000 issues of the daily newspaper *Avui* (`http://www.avui.es`) was used; after removing the *hapax legomena*, about 25000 words and 68000 different prefixes were collected. When a key sequence is entered, it may correspond to one or more words in the dictionary, because of the ambiguity of the keypad: statistics observed are shown in table 2, which shows the occurrence in the corpus of words whose key sequences correspond to a single word, to two words, etc. As may be seen, more than half of the words collected are uniquely defined by a key sequence. However, the fraction of words in the corpus that are correctly reproduced by selecting the *most frequent word* associated to the corresponding keypad sequence raises to 86.0%.[6]

[6]Choosing at random any of the words associated to an ambiguous keypad sequence would have given 51.5% + 13.6%/2 + 24.5%/3 + ... = 68.7%.

AMBIGUITY DEGREE	FREQUENCY
1	51.5%
2	13.6%
3	24.5%
4	6.1%
> 4	5.0%

Table 2: Corpus frequencies for words versus ambiguity degree of their key sequences (the ambiguity degree of a key sequence is the number of different words that map to that sequence).

As in Dunlop and Crossan's work, the choice of corpus biases the dictionary; for example, the names of prominent present-day Catalan politicians are all in the dictionary. It has to be noted also that in these experiments only those Catalan multi-word forms containing apostrophes or hyphens that were actually found in the texts were included in the dictionary, with no attempt whatsoever to generalize (which would be needed in a real application).[7]

4.2 Evaluation of text-entry methods

A comparative evaluation of the relative advantages of different text-entry methods cannot be complete unless one takes in consideration various ergonomical and biomechanical aspects of text entry and the cognitive issues arising (see Dunlop and Crossan (2000)). These are very hard to evaluate unless a physical phone-like interface is built and experiments are conducted with a group of users. Therefore, it has been decided to evaluate only the number of keystrokes necessary to produce each word with each possible text-entry method, and to average this number over electronically available texts.

With the current method, there is more than one way to obtain a given word, because the user may not become aware that the continuation suggested on the screen is the one desired, and may go on keying a

[7]For example, one finds *d'estat* and *l'estat*, but finds *d'amic* and not *l'amic*; one finds *fer-los* i *dir-los* but finds *fer-les* and not *dir-les* (all correct Catalan forms).

longer prefix of the word until the user realizes that the complete key can be used. It will be assumed that the user is always looking at the screen and takes advantage of the suggested continuation (if applicable) as soon as possible (the effect of word completion will be separately evaluated). On the other hand, it will be assumed that the user only presses the * key after having keyed a whole word when the desired result is not on the screen, but never during the process. The rest of the details are as in section 3.2.

Three different text-entry methods have been tested: the traditional multitap method, the predictive text-entry method without word continuation and the predictive text-entry method with word continuation. They have been tested on three texts: the corpus used to build the dictionary (training text), another text from the same newspaper containing 41,000 words, and the following synthetic text containing sentences written "in the style of messages one would send by mobile phone" (Dunlop and Crossan 2000)[8]:

```
Ens pots anar a buscar a l'aeroport?
La meua gossa es diu Xila.
Hola, com va la cosa?
Ei, sóc al bar, on ets tu ara?
Al súper. Que et fa falta res?
He perdut el tren; agafaré el següent.
```

The results are shown in table 3. Three conclusions are clear:

- predictive text entry clearly reduces the number of keystrokes per letter;
- word completion reduces it even further;
- keystroke savings are smaller, but still very distinct, when the texts are not of the same kind as the corpora used to build the probabilistic finite-state dictionary.

It may be argued that a corpus of real text messages would be the best corpus to build a probabilistic dictionary, but a number of problems make it clear that the solution is not so straightforward:

- It is sometimes difficult to identify reliably the language of messages, especially when very similar languages (such as Spanish and Catalan) coexist, because messages are very short. In addition, many adult Catalan speakers have not been taught Catalan in school,[9] therefore do not master the spelling of standard Catalan, and may resort to using Spanish or hybrid ortographical rules to spell the words, which aggravates the problem of language identification.
- A large fraction of messaging traffic contain many abbreviations (some of them comprehensible only to small groups of users), which are used to alleviate the burden of the traditional (multitap) text-entry method; the impressive success of mobile phones among teenagers who use them for messaging much more often than grown-ups may aggravate this. How can one select those abbreviations that should be added to dictionaries?[10]

A local, partial solution to these problems would be having a stochastic finite-state dictionary that is updated by recomputing frequencies for known words and adding words each time the system enters the traditional mode for a particular unknown word[11]

5 Concluding remarks

A method to use text corpora to implement predictive text entry for mobile phone keyboards and for medium-sized languages (such as Catalan) has been presented. The method straightforwardly builds a probabilistic finite automaton representing the dictionary and the frequencies of words and their prefixes and

[8]The sentences are actually a free translation of those in Dunlop and Crossan (2000). This test set is very short and is only used to get an indication of out-of-corpus performance.

[9]Teaching of Catalan in schools resumed in the mid-seventies after four decades of Spanish-only teaching.

[10]Tegic's Spanish *T9* dictionaries already contain many common-use technological abbreviations (such as *FTP* or *HTTP*).

[11]Some word processors such as Sun Microsystems' StarOffice have a contextual word completion feature.

Text	keystrokes per letter		
	traditional	new, no compl.	new, with compl.
Training text	2.019	1.083 (53.6%)	0.898 (44.5%)
More news	2.018	1.116 (55.3%)	0.940 (46.6%)
Synthetic message	1.938	1.347 (69.5%)	1.256 (64.8%)

Table 3: Average number of keystrokes per letter for different texts and different text-entry methods. The percentages in the predictive text-entry methods are relative to the traditional method.

then uses it to almost half the number of keypresses per letter necessary to enter text with respect to the traditional (multitap) method, even when texts entered are different to those used in training.

A number of extensions to this work are envisioned: the use of existing complete Catalan dictionaries containing forms not found in training corpora; a more general treatment of apostrophes and hyphens, and a real user study using an interface as close as possible to that of a real mobile phone.

Acknowledgements: Support from the Spanish *Comisión Interministerial de Ciencia y Tecnología* through grants TIC97-0941 and TIC2000-1599-C02-02 is acknowledged. I also thank Rafael C. Carrasco for valuable comments.

References

Arnott, J. and Javed, M. (1992). Probabilistic character disambiguation for reduced keyboards using small text samples. *Augmentative and Alternative Communication (ISSN 0743-4618)*, 8:215–223.

Carrasco, R. C. and Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In Carrasco, R. and Oncina, J., editors, *Grammatical Inference and Applications*, pages 139–152. Springer-Verlag. Proceedings of the Second International Colloquium on Grammatical Inference, Alicante, Spain, September 1994.

Dunlop, M. and Crossan, A. (1999). Dictionary based text entry method for mobile phones. In *Proc. 2nd Workshop on Human Computer Interaction with mobile devices.*

Dunlop, M. D. and Crossan, A. (2000). Predictive text entry methods for mobile phones. *Personal Technologies*, 4(2):134–143.

Kushler, C. (1998). AAC [augmentative and alternative communication] using a reduced keyboard. In *CSUN 1998 Conference Proceedings.* (available at `http://www.dinf.org/csun_98/csun98_140.htm`).